FIP DEVICE MANAGER NT
DRIVER Multi Boards - Release 2
User Reference Manual

ALS 53328 c-en

First issue: 02-2000
This edition: 01-2001

Meaning of terms that may be used in this document / Notice to readers

WARNING

Warning notices are used to emphasize that hazardous voltages, currents, temperatures, or other conditions that could cause personal injury exist or may be associated with use of a particular equipment.

In situations where inattention could cause either personal injury or damage to equipment, a Warning notice is used.

Caution

Caution notices are used where there is a risk of damage to equipment for example.

Note

Notes merely call attention to information that is especially significant to understanding and operating the equipment.

This document is based on information available at the time of its publication. While efforts have been made to be accurate, the information contained herein does not purport to cover all details or variations in hardware or software, nor to provide for every possible contingency in connection with installation, operation, or maintenance. Features may be described herein which are not present in all systems. ALSTOM assumes no obligation of notice to holders of this document with respect to changes subsequently made.

ALSTOM makes no representation or warranty, expressed, implied, or statutory with respect to, and assumes no responsibility for the accuracy, completeness, sufficiency, or usefulness of the information contained herein. ALSTOM gives no warranties of merchantability or fitness for purpose.

In this publication, no mention is made of rights with respect to trademarks or tradenames that may attach to certain words or signs. The absence of such mention, however, in no way implies there is no protection.

Partial reproduction of this document is authorized, but limited to internal use, for information only and for no commercial purpose.

However, such authorization is granted only on the express condition that any partial copy of the document bears a mention of its property, including the copyright statement.

All rights reserved.
© Copyright 2001. ALSTOM (Paris, France)

Index letter	Date	Nature of revision
a	**02-2000**	Document creation
b	**09-2000**	New: CC141 board Subscriber configuration service Bus Arbiter Service
		Modification of the `kfdm_mps_var_synchronize()` function
c	**01-2001**	Update of the subscriber configuration service

Revisions

1. PURPOSE OF MANUAL AND DOCUMENTED VERSION

This document is the manual for the implementation of FIP DEVICE MANAGER Version 4 destined for a Windows NT 4 environment on PCs equipped with CC138, CC139, CC140 and CC141 PCI boards. This product therefore completes the FIP DEVICE MANAGER offer, within the framework of ALSTOM's WorldFIP technological offer, globally marketed under the FIPWARE trademark.

This document describes:

- installation, configuration and start-up using Windows NT 4's standard driver handling functions,
- the special features of this version as regards the application interface, in relation to the basic interface offered by the FIP DEVICE MANAGER Version 4 function library.

This document in no way constitutes a training manual for the operating concepts and principles of the WorldFIP network.

Users, depending on whether they are installing this software on a PC to serve as a user application, or whether they plan to carry out the interfacing of an application, must have either a working or advanced knowledge of WorldFIP operating principles.

Users who plan to develop the interfacing of a Windows application with this software must follow the FIP DEVICE MANAGER Version 4 manual as it is indispensable to this task.

2. CONTENT OF THIS MANUAL

This user manual is structured in the following way:

Chapter 1: **Introduction:** Description of the software in relation to the hardware and software environment into which it is integrated.

Chapter 2: **Installation:** Commissioning of the software under a Windows environment on a PC equipped with a WorldFIP coupling board (installation, definition of parameters, start-up, stop).

Chapter 3: **Use.**

Chapter 4: **Detailed description of the API:** Description of the programming interface, on the basis of the one described in the FIP DEVICE MANAGER, Version 4 user manual and additional information relating to a Windows environment.

Chapter 5: **Examples of usage.**

The following conventions are used in this manual.

C code sequences are indicated in `courrier` font.

Constants which exist in .h files are indicated in **BOLD** type**.**

3. RELATED PUBLICATIONS

The documents quoted in this manual are specified in the text in square brackets and listed in this paragraph:

- **[1]** FIP DEVICE MANAGER Software Version 4 User Reference Manual ALS 50278
- **[2]** CC138 - CC139 - CC140 - CC141 Board User Manual ALS 53316

For more information on WorldFIP and hardware components:

- **[3]** FIP Network General Introduction ALS 50249
- **[4]** FULLFIP2 User Reference Manual ALS 50262

4. WE WELCOME YOUR COMMENTS AND SUGGESTIONS

ALSTOM strives to produce quality technical documentation. Please take the time to fill in and return the "Reader's Comments" page if you have any remarks or suggestions.

Reader's comments

ALS 53328 c-en	***FIP DEVICE MANAGER NT DRIVER Multi Boards - Release 2 User Reference Manual***

Your main job is:

- ❑ System designer
- ❑ Distributor
- ❑ System integrator
- ❑ Installer
- ❑ Programmer
- ❑ Maintenance
- ❑ Operator
- ❑ Other (specify below)

If you would like a personal reply, please fill in your name and address below:

COMPANY:.....NAME:.....

ADDRESS:.....

.....COUNTRY: .....

Send this form directly to your ALSTOM sales representative or to this address:

ALSTOM Technology
Technical Documentation Department (TDD)
23-25 avenue Morane Saulnier
92364 Meudon la Forêt Cedex
France
Fax: +33 (0)1 46 29 10 21

All comments will be considered by qualified personnel.

REMARKS

Reader's comments

Contents

CHAPTER 1 - INTRODUCTION

1. DESCRIPTION OF THE SOFTWARE....1-1
2. DEVICE DRIVER ARCHITECTURE....1-1

CHAPTER 2 – INSTALLATION

1. SUPPLY....2-1
2. INSTALLATION....2-2
3. BIND A DRIVER TO A BOARD....2-3
4. CONFIGURATION....2-4
5. START-UP / STOP....2-7
   - 5.1 Start-up....2-7
   - 5.2 Stop....2-8
6. SERVICES....2-9
   - 6.1 Installation of the services....2-9
   - 6.2 Configuration of the services....2-10
     - 6.2.1 Subscriber Configuration Service tab....2-11
     - 6.2.2 BA Service tab....2-13
   - 6.3 Start/Stop of the services....2-14
     - 6.3.1 Start....2-15
     - 6.3.2 Stop....2-15

CHAPTER 3 - USE

1. PROGRAMMING INTERFACE....3-1
2. TIME MANAGEMENT....3-4
3. PERFORMANCE....3-5
4. ERROR PROCESSING....3-6
   - 4.1 GetLastError() function....3-6
   - 4.2 Using Windows NT 4 Event Log....3-7
   - 4.3 Using the ErrorLookup utility....3-8
   - 4.4 Display of the last error in a Modal dialog box....3-9
     - 4.4.1 kufdm_format_message()....3-9
     - 4.4.2 kufdm_load_library()....3-10

CHAPTER 4 - DETAILED DESCRIPTION OF THE API

1. kfdm_create_contexte....4-1
2. kfdm_initialize_network....4-2
3. kfdm_valid_medium....4-5
4. kfdm_stop_network....4-6
5. kfdm_ae_le_create()....4-7
6. kfdm_mps_var_create()....4-8
7. kfdm_ae_le_start....4-9
8. kfdm_ae_le_stop()....4-10
9. kfdm_ae_le_delete()....4-11

Contents

10. kfdm_mps_var_write_loc() 4-12
11. kfdm_mps_var_read_loc() 4-13
12. kfdm_mps_var_synchronize() 4-14
13. kfdm_mps_var_read_far() 4-16
14. kfdm_mps_var_write_far() 4-18
15. kfdm_ba_load_macrocycle_manual() 4-20
16. kfdm_ba_load_macrocycle_fipconfb() 4-22
17. kfdm_ba_delete_macrocycle() 4-23
18. kfdm_ba_start() 4-24
19. kfdm_ba_stop() 4-25
20. kfdm_ba_status() 4-26
21. kfdm_ba_commute_macrocycle() 4-27
22. kfdm_ba_set_priority() 4-28
23. kfdm_ba_set_parameters() 4-29
24. kfdm_read_presence() 4-30
25. kfdm_read_present_list() 4-31
26. kfdm_read_report() 4-32
27. kfdm_read_identification() 4-33
28. kfdm_read_ba_synchronize() 4-35
29. kfdm_messaging_fullduplex_create() 4-36
30. kfdm_messaging_to_send_create() 4-38
31. kfdm_messaging_to_rec_create() 4-40
32. kfdm_messaging_delete() 4-41
33. kfdm_channel_create() 4-42
34. kfdm_channel_delete() 4-43
35. kfdm_send_message() 4-44
36. kfdm_received_message() 4-47
37. kfdm_switch_image() 4-49
38. kfdm_get_image() 4-50
39. kfdm_ae_le_get_state() 4-51
40. kfdm_generic_time_initialize 4-52
41. kdfm_generic_time_set_priority 4-53
42. kfdm_generic_time_set_candidate_for_election 4-54
43. kfdm_generic_time_get_election_status() 4-55
44. kfdm_generic_time_delete 4-56
45. kufdm_get_subscriber_number 4-57

CHAPTER 5 - EXAMPLES OF USAGE

GLOSSARY

Figures

Figure 2.1 – FIP Driver NT Configurator....2-4
Figure 2.2 – Subscriber Configuration Service tab....2-11
Figure 2.3 – BA Service tab....2-13

Tables

Table 3.1 – Comparison FDM V4 vs. FDM V4 for Windows NT4 3-3

Chapter 1 Introduction

1. DESCRIPTION OF THE SOFTWARE

This software package, which is part of the FIP DEVICE MANAGER Version 4 functions library, contains a Windows NT 4 driver for WorldFIP boards CC138, CC139, CC140 and CC141 on PCI bus. These boards are PCI boards fitted with the chipset comprising FULLFIP2, FIELDUAL, FIELDRIVE, FIELDTR, etc.

This software package can drive more than one board.

This solution was designed to fulfil a need which is two-fold:

- a need for the simplified implementation of the FIP DEVICE MANAGER Version 4 library under a Windows NT4 environment, requiring only the use of standard driver handling functions (installation, definition of parameters, start-up, stop),
- a need for a level of performance which is very close to that which would be obtained with an equivalent processor in the absence of any operating system.

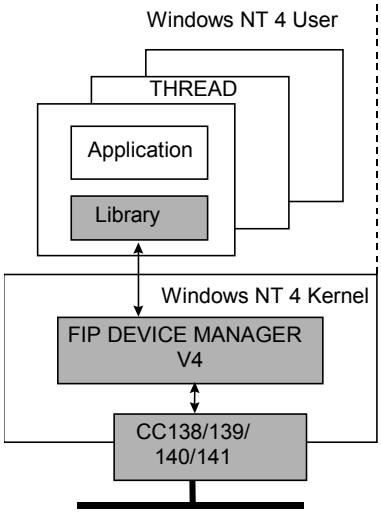
The software has certain limitations such as: the quantity of communication objects (150 VCOMs, 48 AELEs, 2 macrocycles, 40 messaging contexts and 9 messaging system queues) as well as certain other limitations.

2. DEVICE DRIVER ARCHITECTURE

The software is composed of a driver and a programming interface in C/ANSI which provides access to the driver.

The driver integrates the FIP DEVICE MANAGER Version 4 library which provides access to the WorldFIP network for target hardware based on a FULLFIP2 communication coprocessor. Moreover, it manages the range of FDM services, whether for initialisation, timers or the processing of interrupts.

The programming interface (API) which communicates with the driver via the standard functions provided by Windows NT 4 (I/OCTRL, READ/WRITE FILE) provides a means for accessing the driver and, via the driver, for accessing the different functions of FIP DEVICE MANAGER Version 4.



Benefiting therefore from the facilities of a Windows environment, different applications organised in several THREADS may access this driver without the need for any special precautions.

Chapter 2 Installation

1. SUPPLY

The software is provided on a CD-ROM. The directory *Programs* contains the device drivers:

- Fdm_nt_knl_m_pciR2.x: contains the driver for a multi PCI boards.

2. INSTALLATION

The software may only be installed on a PC with the hardware configuration necessary to accomodate NT4, with a PCI V2.1 (or later) slot with a CC138, CC139, CC140 or CC 141 board and a CD-ROM.

The installer must have administrator access rights to install this driver.

Installation procedure:

- insert the CD-ROM into the CD leader,
- change the directory to *\Programs\ Fdm_nt_knl_m_pciR2.x*,
- double-click on the *Setup* icon,
- reboot the computer or the driver will not appear in the list of peripheral devices.

After booting the PC you can verify in

Start → Programs → ALSTOM Technology FIPWARE → FDM NT MULTI

that there are the following 6 sub-items under the *FIP Drivers* item:

- **Researcher of FIP Board**: Tool used for the association of the NT driver with a PCI communication board.
- **Present List**: Tool used to display the present subscribers
- **SMMPS**: Tool used to display the SMMPS variables of a subscriber
- **Error Lookup**: Tool that shows you the details of the errors.
- **FIP Device Manager Doc**: basic FDM document.
- **NT Driver Doc**: this document.

3. BIND A DRIVER TO A BOARD

The **FipBoardConf.exe** tool has to be used for the association of the FIP NT driver to a PCI communication board.

This tool can be launched:

- by double-clicking **FipBoardConf.exe,**
- using the following path:

Start → Programs → ALSTOM Technology FIPWARE → FDM NT MULTI → Researcher of WorldFIP boards.

4. CONFIGURATION

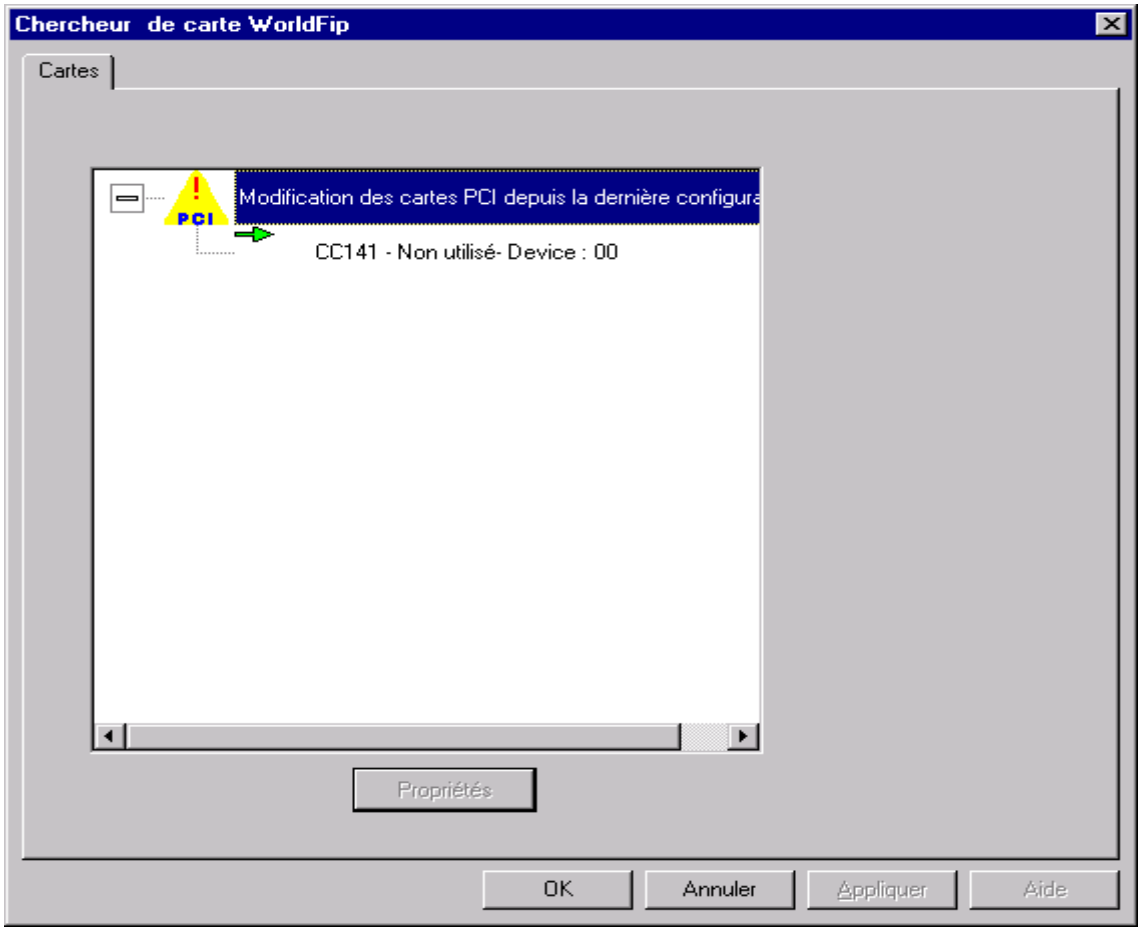
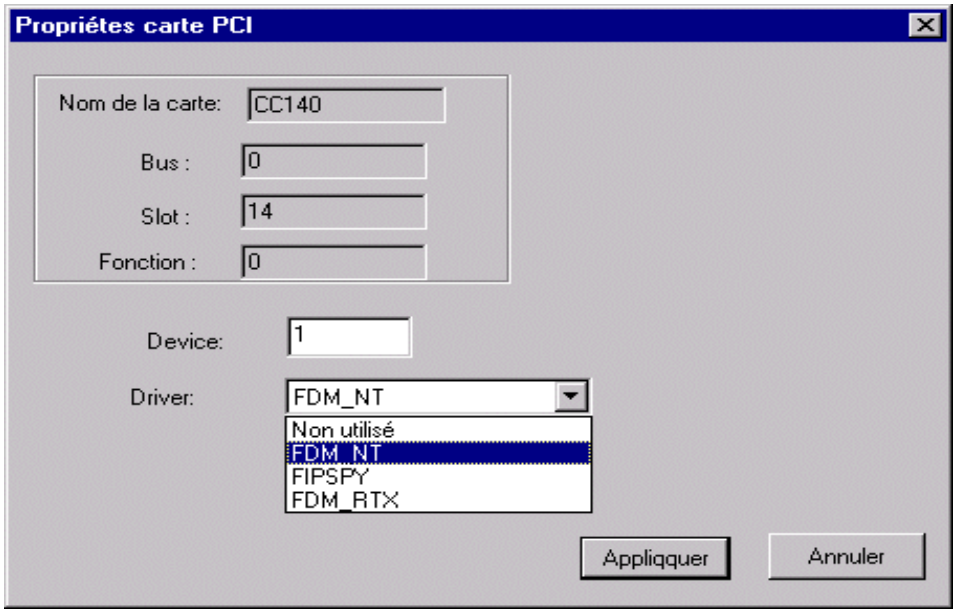


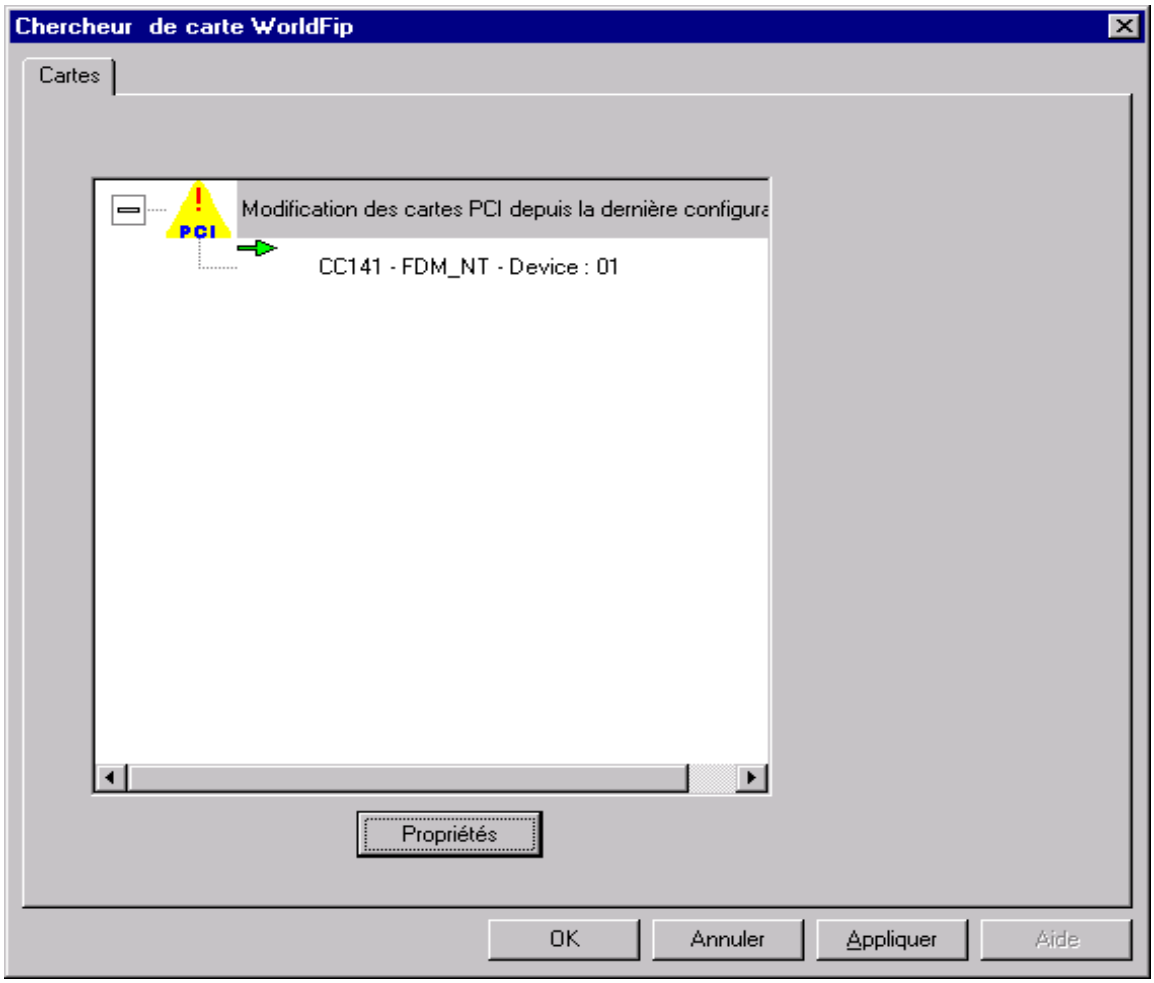
Figure 2.1 – FIP Driver NT Configurator

Select the board and then click on the *Properties* button. The following screen will be displayed:



Select the FDM_NT driver. Enter a device number (1 to 20). This device number associates a the selected board with a driver. Click on *Apply*.

The following screen will be displayed:



This screen shows you the following couple: the board – the associated driver – the device number.

5. START-UP / STOP

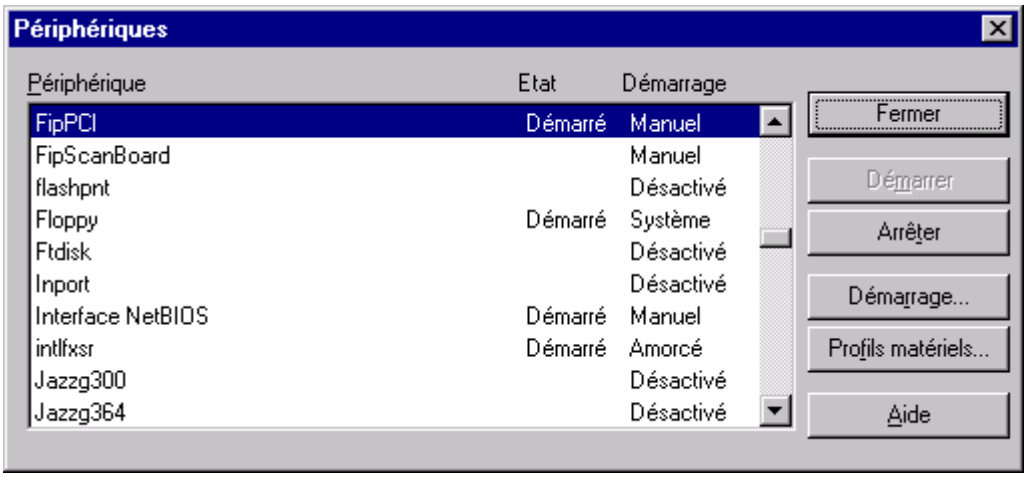
The driver is started up and stopped independently of the user application by using the Windows NT 4 peripheral device management utility.

Directory path:

Start → Settings → Control Panel → Devices.

5.1 Start-up

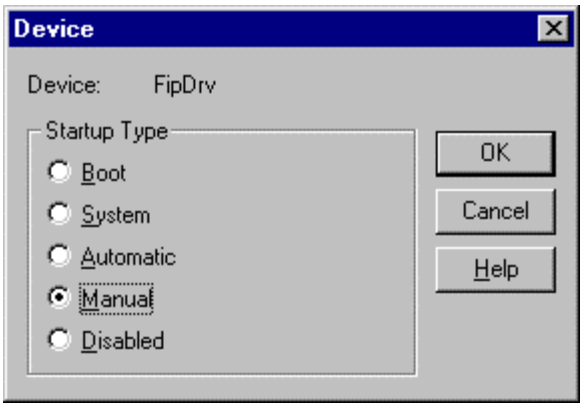
To start-up, select the 'FipPCI' driver and click on *Start*.



The process for starting up the driver includes a phase where the consistency of the values entered for the different parameters is verified. This may produce an error message.

Two types of start-up are available in the menu for starting up peripheral devices and these may be selected in the window which appears by clicking on *Start*.

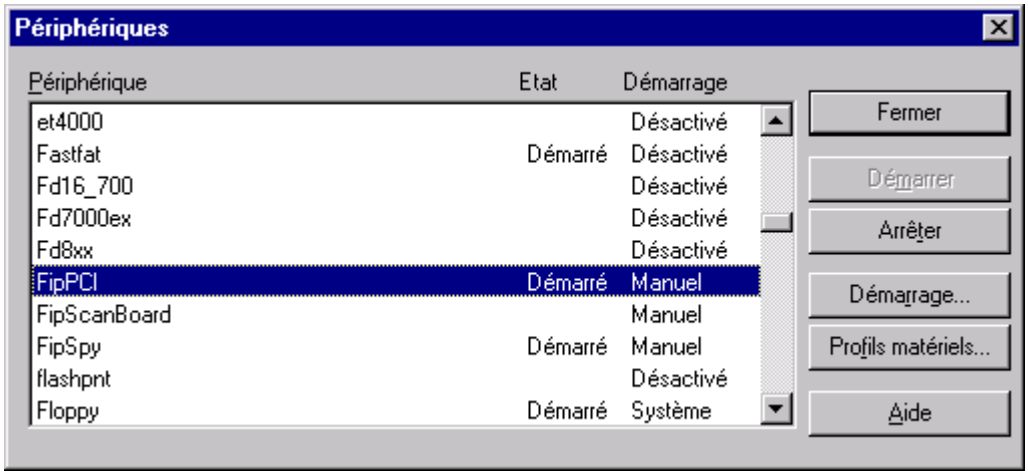
The window below then appears:



The *Manual* start-up option requires that the driver be relaunched following the procedure listed above each time a Windows NT session is opened (e.g.: when the PC is started up), while the *Automatic* start-up option allows the system to be re-booted without intervention.

5.2 Stop

To stop a device select the *FipPCI* driver which is in 'Started' status and click on *Stop*.



6. SERVICES

Two services are available:

- A service that configures the parameters of a subscriber associated with a PCI board
- A service that launches a Bus Arbitrator on a subscriber associated with a PCI board

Note

Before installing the services you have to start the device driver (see Chapter 4.1) and then associate a board with a driver (see Section 3).

6.1 Installation of the services

To install the FDM NT services:

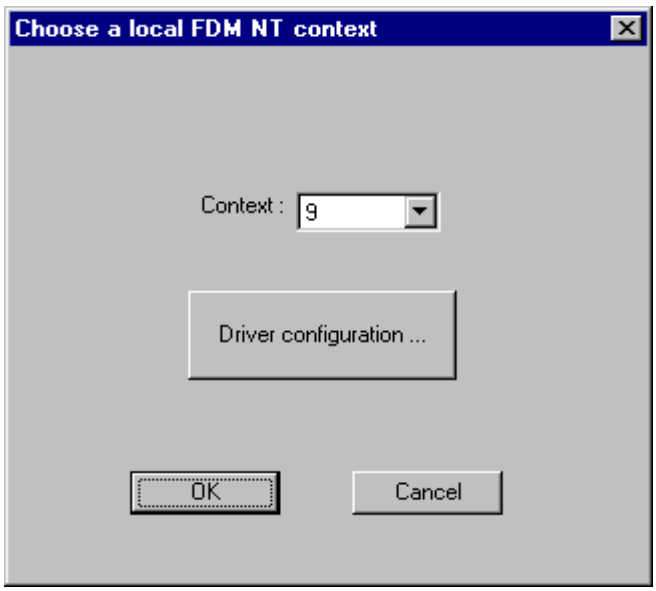
- change the directory to *Programs\Fdm_nt_knl_m_pciR2.x\Services*
- double click the **SETUP.EXE** program.

6.2 Configuration of the services

You can lauch this tool:

- by double-clicking **FDM_NT_CONFIG.EXE**
- using the following path :

Start → Programs → ALSTOM Technology FIPWARE → FDM NT MULTI → Services → FDM_NT_CONFIG



Context:

A combo box displays the created contexts (see Section 3). Select a created context and click OK.

Driver configuration :

If you click this button then the **FipBoardConf.exe** tool is launched (see Section 3). You can consult the couple: board – driver - device or change any of the couple parameters.

6.2.1 Subscriber Configuration Service tab

This tab is used to set the parameters of the subscriber associated with the selected context.

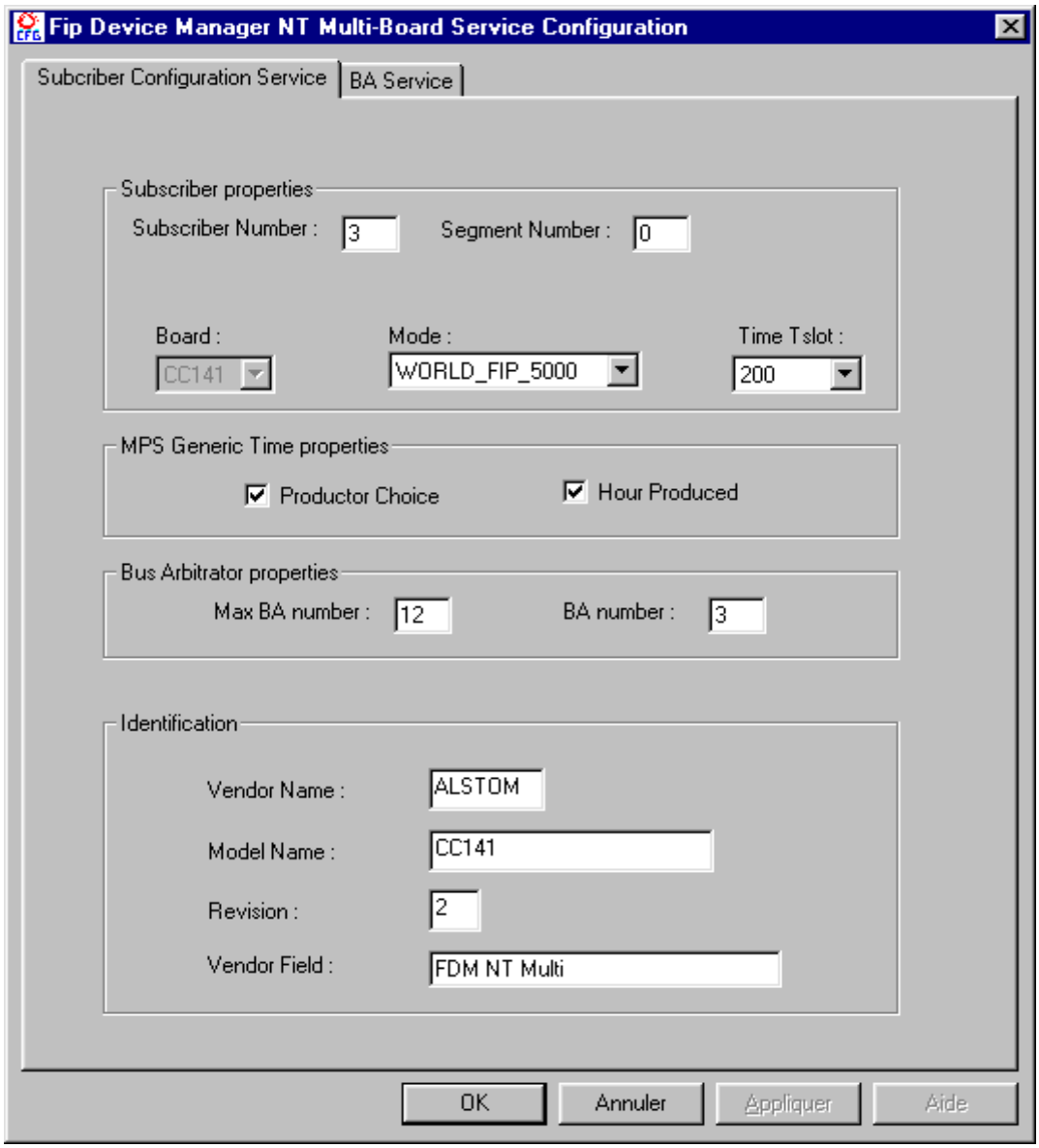


Figure 2.2 – Subscriber Configuration Service tab

Subscriber number:

Subscriber number on the WorldFIP network.

Segment number:

Number of the network segment on which the subscriber is located. This number appears in the messaging system address and its default value is 0.

Board:

Type of the board used. The possible PCI boards are CC138, CC139, CC140 or CC141.

Mode:

See FIP DEVICE MANAGER User Manual ("Mode" parameter of the FDM_CONFIGURATION_SOFT structure in the section that describes the function `fdm_initialize_network()`).

Time Slot:

Value of the time unit used as a basis for all time calculations (see the "Tslot" parameter of the `fdm_initialize_network()` function described in the FDM User Manual).

MPS Generic Time Properties:

Checking the *Hour Produced* option allows you to produce the time (POSIX format, 9802 Identifier, 500 ms refreshing period, Tslot precision). In addition you can select the mechanism for managing the redundancy of the time producer when you check the *Productor Choice* option.

BA Number:

Number of the Bus Arbitrator of your station.

Max. BA Number:

Highest number of the Bus Arbitrators connected on your network.

Vendor Name:

Enter the name of the manufacturer.

Model Name:

Enter the name of the product.

Revision:

Enter the revision index of the product.

Vendor Field:

Enter additional information relating to the product.

6.2.2 BA Service tab

This tab is used to set some parameters if your subscriber has also to act as a BA.

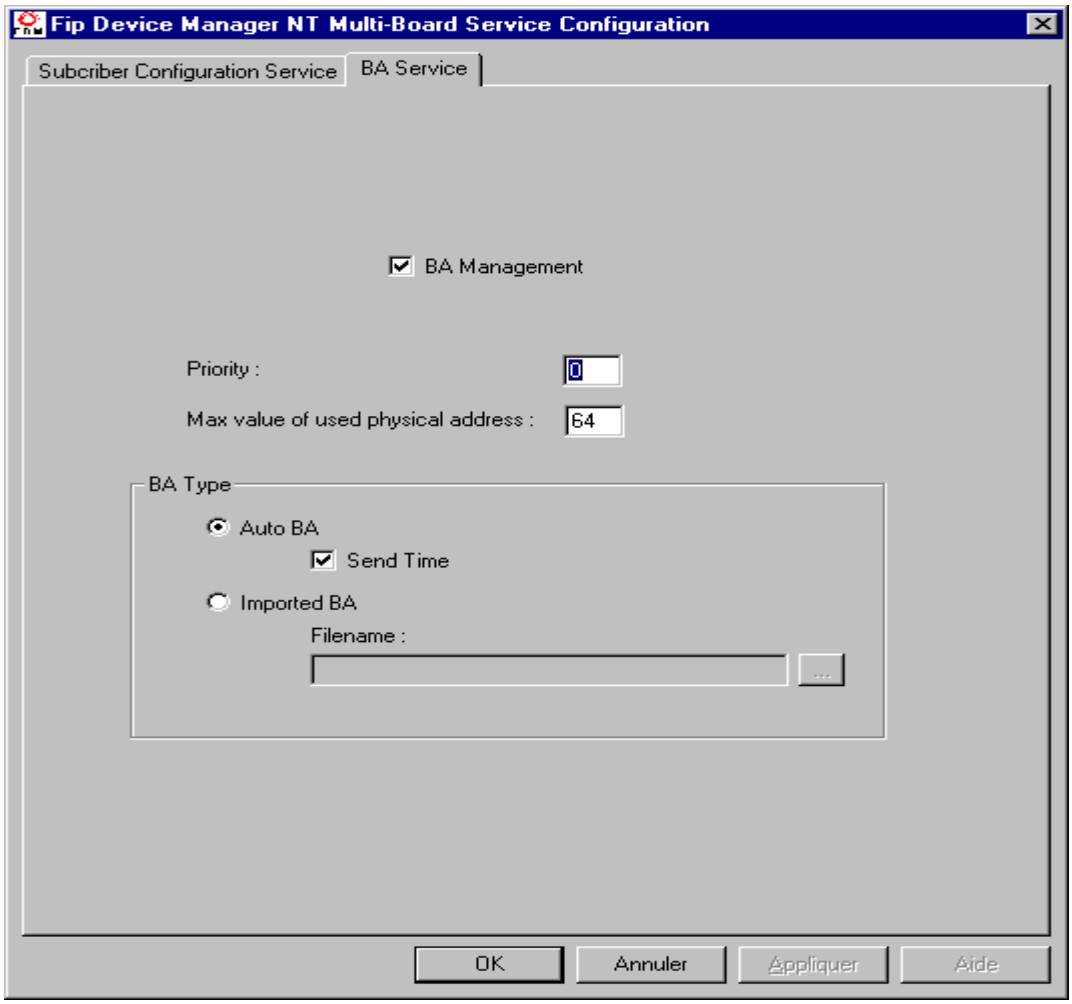


Figure 2.3 – BA Service tab

BA Management:

Check this box, if your subscriber has also to act as a BA.

Priority:

Enter the priority of your BA (between 0 to 15). The highest priority is 0.

Max value of physical address used:

Enter the maximum value of the physical address of the subscribers present on the network.

Auto BA:

If you select this radio button then an *Auto BA* is generated. A periodical messaging list on the identifiers ID_MSG 10XX is created dynamically (XX is the number of a present subscriber detected on the network).

Send Time:

In the case of an *Auto BA*: if you check this box then the identifier ID_DAT 9802 is added to the above list.

Imported BA:

If you select this radio button then the BA is imported from an external tool. In this case you have to enter the name of the binary file.

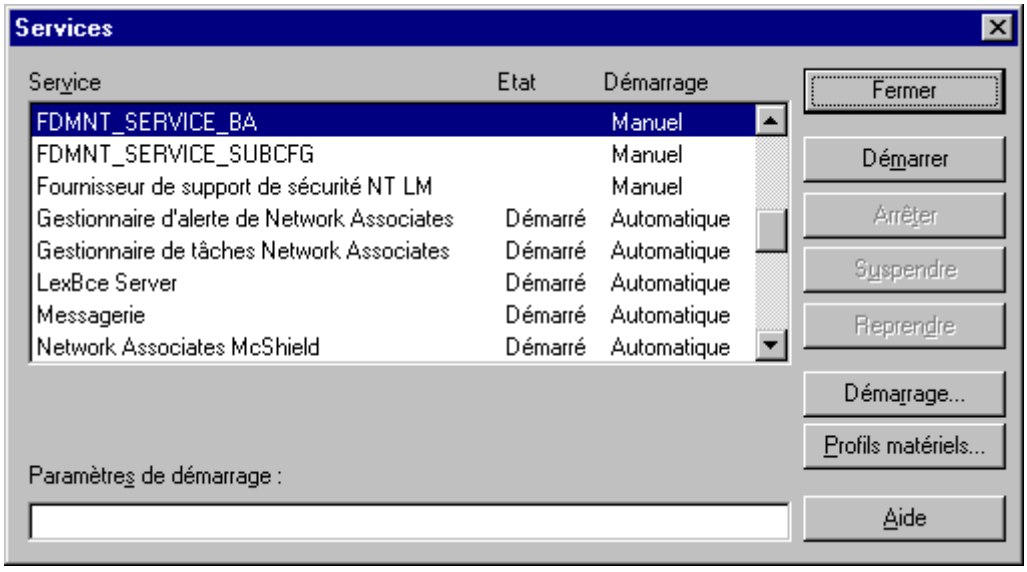
File name:

In the case of an *Imported BA* enter the name of the binary file.

6.3 Start/Stop of the services

Selection path:

Start → *Settings* → *Control Panel* → *Services*



FDMNT_SERVICE_BA: service that manages a bus arbitrator.

FDMNT_SERVICE_SUBCFG: service that initialises the subscriber.

6.3.1 Start

Select the service and then click the *Start* button.

6.3.2 Stop

Select the service and then click the *Stop* button.

Note

If you need the service to start automatically you have to click the *Startup* button and then select the *Automatic* radio button.

Chapter 3 Use

1. PROGRAMMING INTERFACE

The package comes supplied with an API which must be included in the application. This application is compiled using a Visual C++ compiler V6.0 or above.

To do this, the file *kfdm.c* must be included in the project and the following line must be written in the application files which use this API: `# include ''kufdm.h''`

The interface which is then available differs principally from the basic FDM V4 interface in terms of:

- the management by the driver of initialisation functions (hardware, software, network, medium) and the management of communication (ticks, interrupts, etc.),
- services for the exchange of variables, messages and bus arbitration, which are limited to standard services,
- the time management service, which is limited to broadcasting the time of the PC and managing the redundancy of the time producer,
- additional functions, used to create the contexts necessary for Windows NT applications and to manage error processing.

Comparison table:

FIP DEVICE MANAGER V4	API FDM V4 for Windows NT 4
fdm_initialize	*Carried out upon installation of the driver*
fdm_get_version	*Information given in the configuration splash screen*
fdm_ticks_counter	*Run periodically by the driver, every 500 ms*
fdm_change_test_medium_ticks	*No such procedure*
fdm_initialize_network	*kfdm-initialize-network*
fdm_stop_network	*kfdm-stop-network*
fdm_valid_medium	*kfdm-valid-medium*
fdm_online_test	*No such procedure*
fdm_process_its_fip	*Management of IRQ and EOC is ensured by the driver.*
fdm_process_it-eoc	
fdm_process_it_irq	
	kfdm_create_contexte
fdm_ae_le_create	kfdm_ae_le_create
fdm_ae_le_delete	kfdm_ae_le_delete
fdm_ae_le_start	kfdm_ae_le_start
fdm_ae_le_stop	kfdm_ae_le_stop
fdm_ae_le_get_state	kfdm_ae_le_get_state
fdm_mps_var_create	kfdm_mps_var_create
fdm_mps_var_change_id	
fdm_mps_var_change_period	
fdm_mps_var_change_Rqa	*On-line modification of parameters cannot be used.*
fdm_mps_var_change_MSGa	
fdm_mps_var_change_priority	
fdm_mps_var_change_prod_cons	
fdm_mps_var_write_loc	kfdm_mps_var_write_loc
fdm_mps_var_read_loc	kfdm_mps_var_read_loc
fdm_mps_var_time_write_loc	*No such procedure exists*
fdm_mps_var_time_read_loc	*No such procedure exists*
fdm_mps_var_write_universal	kfdm_mps_var_write_far
fdm_mps_var_read_universal	kfdm_mps_var_read_far
fdm_mps_fifo_empty	*Not used*
fdm_generic_time_initialize	*Kfdm-generic-time-initialize*
fdm_generic_time_read_loc	*No time consumption possible*
fdm_generic_time_write_loc	*Automatic writing by the driver*
fdm_generic_time_set_priority	kfdm_generic_time_set_priority
fdm_generic_time_set_candidate_for_election	kfdm_generic_time_set_candidate_for_election
fdm-generic-time-delete	Kfdm-generic-time-delete
	kfdm_generic_time_get_election_status

FIP DEVICE MANAGER V4	API FDM V4 for Windows NT 4
fdm_channel_create	kfdm_channel_create
fdm_channel_delete	kfdm_channel_delete
fdm_change_channel_nr	*No such procedure exists*
fdm_messaging_fullduplex_create	kfdm_messaging_fullduplex_create
fdm_messaging_to_send_create	kfdm_messaging_to_send_create
fdm_messaging_to_rec_create	kfdm_messaging_to_rec_create
fdm_messaging_delete	kfdm_messaging_delete
fdm_send_message	kfdm_send_message
fdm_msg_ref_buffer_free	*Not used*
fdm_msg_data_buffer_free	*Not used*
fdm_msg_rec_fifo_empty	*Not used*
fdm_msg_send_fifo_empty	*Not used*
	kfdm_receive_message
fdm_ba_load_macrocycle_fipconfb	kfdm_ba_load_macrocycle_fipconfb
fdm_ba_load_macrocycle_manual	kfdm_ba_load_macrocycle_manual
fdm_ba_delete_macrocycle	kfdm_ba_delete_macrocycle
fdm_ba_start	kfdm_ba_start
fdm_ba_external_resync	*No such procedure exists*
fdm_ba_commute_macrocycle	kfdm_ba_commute_macrocycle
fdm_ba_set_priority	kfdm_ba_set_priority
fdm_ba_set_parameters	kfdm_ba_set_parameters
fdm_ba_status	kfdm_ba_status
fdm_ba_stop	kfdm_ba_stop
fdm_ba_loaded	*No such procedure exists*
fdm_read_report	kfdm_read_report
fdm_read_present_list	kfdm_read_present_list
fdm_read_identification	kfdm_read_identification
fdm_read_presence	kfdm_read_presence
fdm_read_ba_synchronize	kfdm_read_ba_synchronize
fdm_get_local_report	*No such procedure exists*
fdm_switch_image	kfdm_switch_image
fdm_get_image	kfdm_get_image
fdm_smmps_fifo_empty	*Not used*
	kufdm_get_subscriber_number
	kufdm_format_message
	kufdm-load-library

Table 3.1 – Comparison FDM V4 vs. FDM V4 for Windows NT4

Note

A driver is an independent program. If, for example, the application program creates an AE_LE (with `kdfm_ae_le_create()`) in the driver, the AE_LE still exists even when the user application is stopped.

If the user program is relaunched, and an AELE is created with the same row, the driver returns a warning.

This also applies for other objects (BA, messaging context).

2. TIME MANAGEMENT

Depending on the settings made, the PC may choose to broadcast its time over the network thanks to the ID9802 variable and using specific FDM V4 mechanisms. The time is displayed in POSIX format.

The variable produced is updated automatically every 500 ms, with a guaranteed precision of 100 ns for the recopy operation, the time being broadcast over the network using a level of precision fixed by `Time_slot` (selected during configuration).

Note

Therefore, under general conditions of use at 1 Mbit/s, the difference between the time of the PC and the time broadcast will be less than 62.5 µs if `Time_slot` has been selected with this value.

The management of time producer election may be activated on initialisation and is used to select a time producer from amongst various devices that are capable of time production on the same network. Channel 1 must be configured before the election mechanism will work.

3. PERFORMANCE

Performance in terms of device exchange capacity has been defined as follows:

- Medium redundancy: **TWO_BUS_MODE**
- Data base: **TWO_IMAGE_MODE**
- Medium test period: according to the basic FDM document **[1]** (see Chapter 2, Subsection 1.2).
- Value of Time-out: 1 sec
- Value of Time-out message: according to the basic FDM document **[1]** (see Chapter 2, Subsection 1.2).
- Period of on-line tests: not selected (there are no on-line tests)
- BA_DIM dimension: 0x2000
- Maximum number of VCOMs 150
- Maximum number of AELE: 48
- Maximum number of macrocycles: 2
- Maximum number of msg contexts: 40
- Available messaging channels: 0 for aperiodic
  1..8 for periodic

In addition, the application using this API must be compiled in "struct member alignment: 8 bytes" mode, which is the default option proposed by the Visual C++ V6.0 compiler or later.

4. ERROR PROCESSING

There are four methods for managing errors:

- getting the last error by `GetLastError()` function,
- using the NT4 Event Log,
- using the ErrorLookup utility,
- using a Modal dialog box to display the last error.

4.1 GetLastError() function

This function returns the value of the last error code to the calling Thread. The FDM driver, however, updates the error code when a fault occurs.

To facilitate debugging, upon each call to the `SIGNAL_WARNING()` function, which is activated when an error is detected by the FDM, the driver updates the error code accessible by the application and activates the PC buzzer for 5 seconds.

Function:

```
DWORD GetLastError(VOID)
```

Input parameters:

None

Report:

This function returns the error code *dwErrCode*, which is updated during the call for help to the `VOID SetLastError(DWORD dwErrCode)` function.

Error codes are structured in the following way:

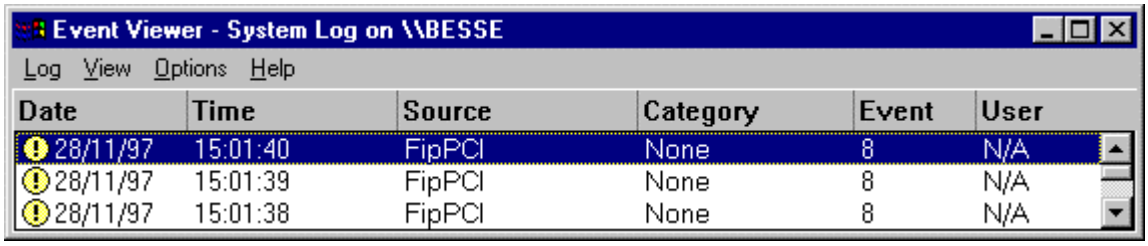
- when the most significant bit (b31) is set to 1, the low order bits represent the errors usually fed back by FDM, as specified by enum CODE_ERROR in its reference manual **[1],**
- 0xD - - - , refers to errors detected by the driver access interface,
- 0xE - - - , refers to errors detected by the driver.

4.2 Using Windows NT 4 Event Log

The event log is used to access a chronological event report.

It is activated by the command:

Start → Programs → Administrative Tools (Common) → Event Viewer


Event Viewer - System Log on \\BESSE
Log View Options Help
Date Time Source Category Event User
28/11/97 15:01:40 FipPCI None 8 N/A
28/11/97 15:01:39 FipPCI None 8 N/A
28/11/97 15:01:38 FipPCI None 8 N/A

Detailed information on a particular event is obtained by double-clicking on it:

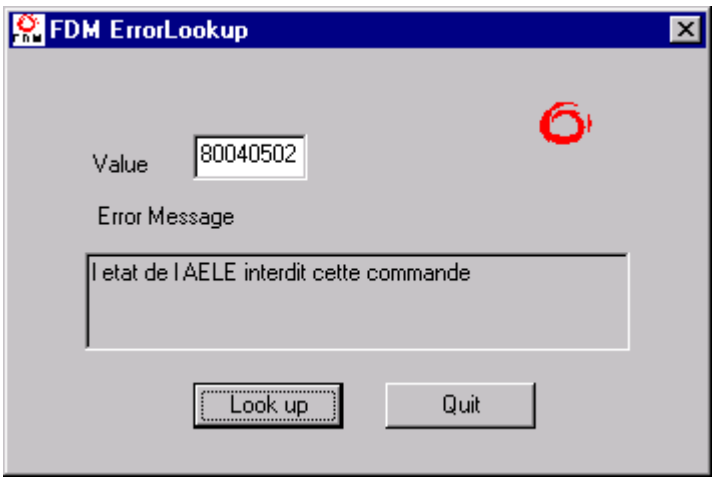
The code contained in the last word is the FDM warning code (0x502 in the example above). Refer to **[1]** for details of all the possible codes.

4.3 Using the ErrorLookup utility

This tool can be launched by:

- double-clicking **ErrorLookup.exe**
- using the following path:

Start→Programs→ALSTOM Technology FIPWARE→FDM NT MULTI→Error Lookup



Enter the error value returned by the driver. The description of the error will be displayed.

4.4 Display of the last error in a Modal dialog box

To display the last error in a Modal dialog box you have to use the following functions:

- `kufdm_format_message()`
- `kufdm_load_library()`

4.4.1 kufdm_format_message()

Corresponds to FDM function:

None.

Prototype:

```
void kufdm_format_message (HINSTANCE hlib ) ;
```

Description:

Utility for displaying the last error in a Modal dialog box. NT errors will be displayed in the language used by the PC.

The error messages are contained in a DLL with the name messages.dll.

To access an error message, you have to load first this DLL by calling the **`kufdm_load_library()`** function.

Note

Your process will be blocked till you click on the OK button of the Modal dialog box.

Report:

Not applicable.

4.4.2 kufdm_load_library()

Corresponds to FDM function:

None.

Prototype:

```
HINSTANCE   kufdm_load_library ( void );
```

Description:

Loads the DLL that contains the error messages used by the function `kufdm_format_message()`

Input parameters:

None.

Report:

Handle on the DLL if the load operation has been performed correctly.

NULL if the load operation has not been performed correctly.

Chapter 4 Detailed description of the API

A detailed description of the programming interface may be found in the reference manual **[1]**. This manual should be used as a reference guide.

1. kfdm_create_contexte

Corresponds to FDM function:

None.

Prototype:

```
HANDLE kfdm_create_contexte ( int Device)
```

Description:

Used to create a logical link between the user world and the kernel world of the FDM driver.

Input parameters:

Device: logical number (1 to 20) that identifies an association of driver to a board and to a driver. Enter the same value that you entered when you used the configuration tool (Chapter 2, Subsection 4, Configuration).

Report:

HANDLE type reference on the open device.

If **INVALID_HANDLE_VALUE** is displayed, opening has not been performed correctly. If it is not produced, the reference of the open device appears.

2. kfdm_initialize_network

Corresponds to the FDM function:

`fdm_initialize_network()`

Prototype:

```
BOOL    kfdm_initialize_network (
        HANDLE                              hdevice ,
        const FDM_CONFIGURATION_SOFT        *  U_soft,
        FDM_CONFIGURATION_HARD              *  U_hard,
        const FDM_IDENTIFICATION            *  U_ident );
```

Description:

Used to initialise the network variables.

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`U_soft:` pointer to FDM_CONFIGURATION_SOFT type data structure described below. The scored out fields are not to be initialised.

```
typedef struct {
      Ushort      Type; //  MESSAGE_RECEPTION_AUTHORIZED
                           | TWO_BUS_MODE
                           | TWO_IMAGE_MODE


      enum        _FULLFIP_Mode_List Mode;
      Ushort      TSlot ;
      Ushort      NB_OF_USER_MPS_VARIABLE ;          // 150
      Ushort      BA_Dim ;                           // 0x2000
      Ulong       FULLFIP_RAM_Dim ;                  // 0x20000
      Ushort      NB_OF_DIFFERENT_ID_PROC_BA ;       // 0x100
      Ushort      Nr_Of_Repeat ;                     // 1
      Ushort      Nr_Of_Tx_Buffer[9] ;               // 4*9
      void    (*User_Present_List_Prog )             // interne
            ( struct _FDM_REF* , FDM_PRESENT_LIST    * ) ;
      Ushort (*User_Identification_Prog )            // interne
            ( struct _FDM_REF* , FDM_IDENT_VAR       * ) ;
      Ushort (*User_Report_Prog )                    // interne
            ( struct _FDM_REF* , FDM_REPORT_VAR      * ) ;
      Ushort (*User_Presence_Prog )                  // interne
            ( struct _FDM_REF* , FDM_PRESENCE_VAR    * ) ;
      void    (*User_Synchro_BA_Prog )               // interne
            ( struct _FDM_REF* , FDM_SYNCHRO_BA_VAR * ) ;

      Ushort      Test_Medium_Ticks;                 // 2*N
      Ushort      Time_Out_Ticks;                    // 4*N
      Ushort      Time_Out_Msg_Ticks;                // 4*N
      Ushort      Online_Tests_Ticks;                // 0

      Ushort      Default_Medium_threshold;          // 2%

      #if   ( FDM_WITH_BA == YES ) && (FDM_WITH_FIPIO == YES
                                          // FDM_WITH_FIPIO = NO
```

```
        Ushort      Segment_Paramers_Ticks;
        #endif
        struct {
        Ushort TIMER_CNT_REGISTER ;                          // 0
        Ushort MODE_REGISTER ;                               // 0
        } User_responsability ;

        void (*User_Signal_Mps_Aper)
                        ( struct _FDM_REF*  ) ;              // interne
        void (*User_Signal_Smmps )
                        ( struct _FDM_REF*  ) ;              // interne
        void (*User_Signal_Send_Msg)
                        ( struct _FDM_REF*  ) ;              // interne
        void (*User_Signal_Rec_Msg )
                        ( struct _FDM_REF*  ) ;              // interne

        void * User_Ctxt;                                    // interne

} FDM_CONFIGURATION_SOFT;
```

U_hard: pointer to **FDM_CONFIGURATION_HARD** type data structure described below. The scored out fields are not to be initialised.

```
typedef struct _FDM_CONFIGURATION_HARD {
      Uchar       K_PHYADR ;
      char        MySegment ;
      Uchar       Reserved[2] ;
#if         ( FDM_WITH_CHAMP_IO == YES )
      __Port_Type__ LOC_FIP[8];                              // interne
      __Port_Type__ LOC_FIPDRIVE[4];                         // interne
#else
      Uchar volatile *  LOC_FIP[8];                          // interne
      Uchar volatile *  LOC_FIPDRIVE[4];                     // interne
#endif
      Ushort volatile *  FREE_ACCES_ADDRESS;                 // interne

      void (*User_Reset_Component)                           // interne
            ( struct _FDM_CONFIGURATION_HARD * );

      void (*User_Signal_Fatal_Error)                        // interne
            ( struct _FDM_REF * Ref ,FDM_ERROR_CODE );
      void (*User_Signal_Warning )                           // interne
            ( struct _FDM_REF * Ref ,FDM_ERROR_CODE );

      MEMORY_RN  *Memory_Management_Ref   ;                  // interne

      struct _FDM_REF *Ptr_Autotests ;                       // interne

 } FDM_CONFIGURATION_HARD ;
```

U_ident: pointer to **FDM_CONFIGURATION_HARD** type data structure described below. The scored out fields are not to be initialised.

```
typedef struct {
      char * Vendor_Name;
      char * Model_Name;
      char * Revision;
      char * Tag_Name;
      char * SM_MPS_Conform;
      char * SMS_Conform;
      char * PMDP_Conform;
      char * Vendor_Field;
} FDM_IDENTIFICATION;
```

Report:

This function returns a report with the boolean value:

TRUE: if the request has been processed correctly.

FALSE: if it has not.

3. kfdm_valid_medium

Corresponds to the FDM function:

`fdm_valid_medium()`

Prototype:

```
BOOL   kfdm_valid_medium   (
       HANDLE hdevice ,
       enum  _MEDIUM_DEF  medium );
```

Description:

Choice of mediums to be used.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`medium:`	medium

Report:

This function returns a report with the Boolean value:

TRUE: if the request has been processed correctly.

FALSE: if it has not.

4. kfdm_stop_network

Corresponds to the FDM function:

`fdm_stop_network()`

Prototype:

```
BOOL   kfdm_stop_network (HANDLE  hdevice );
```

Description:

Used to stop a network.

Input parameters:

`hdevice:` HANDLE type reference on the open device.

Report:

This function returns a report with the Boolean value:

TRUE: if the request has been processed correctly.

FALSE: if it has not.

5. kfdm_ae_le_create()

Corresponds to FDM function:

`fdm_ae_le_create()`

Prototype:

```
BOOL   kfdm_ae_le_create (
       HANDLE      hdevice,
       int         AE_LE_RANG,
       int         AE_LE_DIM);
```

Description:

Creation of an AE_LE.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`AE_LE_RANG:`	row of the AE_LE. Value [0..9].
`AE_LE_DIM:`	maximum number of variables which may make up this AE_LE.

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

6. kfdm_mps_var_create()

Corresponds to FDM function:

`fdm_mps_var_create()`

Prototype:

```
BOOL   kfdm_mps_var_create (
       HANDLE      hdevice ,
       int         AE_LE_RANG ,
       FDM_XAE *   Var_Param );
```

Description:

Defines the parameters of a variable in an AE_LE.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`AE_LE_RANG:`	row of the AE_LE. Value [0..9].
`Var_Param:`	pointer to a structure describing the parameters of the variable.

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

7. kfdm_ae_le_start

Corresponds to the FDM function:

`fdm_ae_le_start()`

Prototype:

```
BOOL   kfdm_ae_le_start (
       HANDLE      hdevice ,
       int         AE_LE_RANG );
```

Description:

Start-up of an AE_LE.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`AE_LE_RANG:`	row of the AE_LE. Value [0..9].

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

8. kfdm_ae_le_stop()

Corresponds to the FDM function:

`fdm_ae_le_stop()`

Prototype:

```
BOOL    kfdm_ae_le_stop (
        HANDLE      hdevice ,
        int         AE_LE_RANG );
```

Description:

Stopping an AE_LE

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`AE_LE_RANG:` row of the AE_LE. Value [0..9].

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

9. kfdm_ae_le_delete()

Corresponds to the FDM function:

`fdm_ae_le_delete()`

Prototype:

```
BOOL   kfdm_ae_le_delete (
       HANDLE      hdevice ,
       int         AE_LE_RANG );
```

Description:

Deletion/destruction of an AE_LE.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`AE_LE_RANG:`	row of the AE_LE. Value [0..9].

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

10. kfdm_mps_var_write_loc()

Corresponds to FDM function:

`fdm_ae_le_delete()`

Prototype:

```
BOOL   kfdm_mps_var_write_loc (
       HANDLE                    hdevice ,
       int                       AE_LE_RANG ,
       int                       VAR_RANG ,
       int                       Lg,
       USER_BUFFER_TO_READ       Data_Buffer) ;
```

Description:

Writing of an MPS variable.

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`AE_LE_RANG:` row of the AE_LE. Value [0..9].

`VAR_RANG:` row of the variable in the AE_LE.

`Lg:` number of bytes of the variable.

`Data_Buffer:` pointer to a USER_BUFFER_TO_READ type structure containing the data to be written. Refer to **[1]** for the description of this structure.

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

11. kfdm_mps_var_read_loc()

Corresponds to FDM function:

`fdm_ae_le_delete()`

Prototype:

```
BOOL   kfdm_mps_var_read_loc (
       HANDLE                    hdevice ,
       int                       AE_LE_RANG ,
       int                       VAR_RANG ,
       K_VAR_DATA  *Data_Buffer) ;
```

Description:

Reads a variable.

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`AE_LE_RANG:` row of the AE_LE. Value [0..9].

`VAR_RANG:` row of the variable in the AE_LE.

`Lg:` number of bytes of the variable.

`Data_Buffer:` pointer to a K_VAR_DATA type buffer described below which will contain the value of the variable read.

```
typedef struct {

      int                       Last_Error;

      FDM_MPS_READ_STATUS       Status;

      FDM_MPS_VAR_DATA          Data;

} K_VAR_DATA;
```

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

12. kfdm_mps_var_synchronize()

Corresponds to FDM function:

None.

Prototype:

```
BOOL    kfdm_mps_var_synchronize (
        HANDLE                              hdevice,
        LPKFDM_ABSTRACT_SYNCHRO_TYPE        InternalInfos,
        LPOVERLAPPED_COMPLETION_ROUTINE     CallBackRoutine);
```

Description:

Enables synchronisation of the application on receipt of the Sync variable declared in one of the AE_LE which have been started up.

Note

This procedure must be "MONO THREAD".

This procedure uses [Write/Read]FileEx in Windows NT.

It must be followed by SleepEx or another equivalent procedure (refer to Microsoft SDK documentation).

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`InternalInfos:` Internal LPKFDM_ABSTRACT_SYNCHRO_TYPE type structure.

The structure must not be modified during execution of the function.

`CallBackRoutine:` Procedure called on receipt of the sync.

The prototype of the `CallBackRoutine()` function is as follows:

```
VOID WINAPI CallBackRoutine  (
                  DWORD                     dwErrorCode,
                  DWORD                     Compte_rendu,
                  LPOVERLAPPED              lpOverlapped     ) ;
```

`dwErrorCode:` Will be equal to 0 if the request has been processed correctly by NT.

`Compte_rendu:` 2: If Sync correct.

3: If Time-out or invalid.

4: Syncho lost

`LpOverlapped:` pointer on the `InternalInfos` structure provided during the call.

`InternalInfos.gOverlapped.IntervalHigh:`

2 => received sync event.

3 => non-received sync event (the time_out is 5 seconds).

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

Example of usage:

```
VOID WINAPI IOCompletionRoutine1(
DWORD  dwErrorCode,              //completion code
DWORD  Compte_rendu,
LPOVERLAPPED                     lpOverlapped  //pointer to structure with
                                 I/O information) ;
{
switch (compte-rendu) {
     case 2:
            PulseEvent (Hevent); //OK
            break;
     case 3:
            // time out
     case 4
            // synchronisation loss
     break;
}
                   ////////////////
main(){
        for ( ; ;)
        {
        KFDM_ABSTRACT_SYNCHRO_TYPE Tmp;
        DWORD Dw ;
        kfdm_mps_var_synchronize (
                    hdevice ,
                    &Tmp ,
                    IOCompletionRoutine );

        Dw = SleepEx (  6000 , THRUE  ) ;
        if ( Dw  == WAIT_IO_COMPLETION ) {
        ----------------→i/o - completion routine OK
        } else {
        ---------------→ i/o - completion routine NOK.
        }
        }
```

13. kfdm_mps_var_read_far()

Corresponds to FDM function:

`fdm_mps_var_read_universal()`

Prototype:

```
BOOL    kfdm_mps_var_read_far (
        HANDLE                              hdevice,
        int                                 AE_LE_RANG,
        int                                 VAR_RANG,
        K_FDM_MPS_VAR_DATA_FAR              *Data_Buffer,
        LPOVERLAPPED                        InternalInfos,
        LPOVERLAPPED_COMPLETION_ROUTINE     CallBackRoutine);
```

Description:

Enables remote reading of an MPS variable.

Note

This procedure uses [Write/Read]FileEx in Windows NT.

It must be followed by SleepEx or another equivalent procedure (refer to Microsoft SDK documentation).

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`AE_LE_RANG:` row of the AE_LE. Value [0..9].

`VAR_RANG:` row of the variable in the AE_LE.

`Data_Buffer:` pointer to a K_FDM_MPS_VAR_DATA_FAR type data structure described below:

```
typedef struct {
     int                    Last_Error;
     FDM_MPS_READ_STATUS    Status;
     FDM_MPS_VAR_DATA       Data;
} K_FDM_MPS_VAR_DATA_FAR;
```

`InternalInfos:` Internal structure.

The structure must not be modified during the execution of the function.

`CallBackRoutine:` Procedure called on receipt of the variable or when time delay expires.

The prototype of the `callBack` function is as follows:

```
VOID WINAPI CallBackRoutine   (
     DWORD              dwErrorCode,
     DWORD              FdmReport,
     LPOVERLAPPED       lpOverlapped     ) ;
```

`dwErrorCode:` is equal to 0 if the request has been processed correctly.

`FdmReport:` = 0 if request is processed by OK FDM.

≠ 0 if NOK.

`lpOverlapped:` pointer to an OVERLAPPED type structure.

The OVERLAPPED structure contains information used in asynchronous input and output (I/O).

```
typedef struct _OVERLAPPED { // o
     DWORD  Internal;
     DWORD  InternalHigh;
     DWORD  Offset;
     DWORD  OffsetHigh;
     HANDLE hEvent;
} OVERLAPPED;
```

Note

The field hEvent can be used as a user tag (see the example).

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

14. kfdm_mps_var_write_far()

Corresponds to FDM function:

`fdm_mps_var_write_universal()`

Prototype:

```
BOOL   kfdm_mps_var_write_far (
       HANDLE                              hdevice ,
       int                                 AE_LE_RANG ,
       int                                 VAR_RANG ,
       int                                 DataLength,
       USER_BUFFER_TO_READ                 DataBuffer,
       LPOVERLAPPED                        InternalInfos ,
       LPOVERLAPPED_COMPLETION_ROUTINE     CallBackRoutine);
```

Description:

Enables remote writing of an MPS variable.

Note

This procedure uses [Write/Read]FileEx in Windows NT.

It must be followed by SleepEx or another equivalent procedure (refer to Microsoft SDK documentation).

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`AE_LE_RANG:` row of the AE_LE. Value [0..9].

`VAR_RANG:` row of the variable in the AE_LE.

`DataLength:` number of bytes to be transferred.

`DataBuffer:` pointer to a USER_BUFFER_TO_READ type data structure of the data to be written. Refer to **[1]** for the description of this structure.

`InternalInfos:` internal structure.

The structure must not be modified during the execution of the function.

`CallBackRoutine:` Procedure called on receipt of the variable or when time delay expires.

The prototype of the `callBack` function is the same as that for the `kfdm_mps_var_read_far()` function.

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

15. kfdm_ba_load_macrocycle_manual()

Corresponds to FDM function:

`fdm_ba_load_macrocycle_manual()`

Prototype:

```
BOOL   kfdm_ba_load_macrocycle_manual (
       HANDLE      hdevice ,
       int                        BA_RANG ,
       int                        Nb_of_Liste,
       int                        Nb_of_Instruction ,
       unsigned short             Label,
       const PTR_LISTS* ,
       const PTR_INSTRUCTIONS *
       );
```

The macro below enables a simplified call to be made:

```
BOOL Bus_Arbitrator__CREATE(
       hdevice,
       BA_RANG,
       Label,
       Listes,
       Programme);
```

Description:

Used to create and load a macrocycle.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`BA_RANG:`	row of the macrocycle [0 .. 1].
`Nb_of_Liste:`	refer to FDM V4 User Manual **[1]**.
`Nb_of_Instruction:`	‘’
`Label:`	‘’
`PTR_LISTS*:`	‘’
`PTR_INSTRUCTIONS *:`	‘’

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

Example of usage:

```
const LIST_ELEMENT L0[] = {
       0xE100 , ID_DAT

};
const LIST_ELEMENT L1[] = {
       0x0001 , ID_DAT  ,
       0x0002 , ID_DAT  ,
       0x0002 , ID_DAT  ,
       0x0003 , ID_DAT  ,
       0x0016 , ID_DAT  ,
       0x0017 , ID_DAT  ,
       0x0018 , ID_DAT  ,
       0x0019 , ID_DAT  ,
       0x0021 , ID_DAT  ,
       0x0022 , ID_DAT  ,
       0x0023 , ID_DAT  ,
       0x0024 , ID_DAT  ,
       0x0025 , ID_DAT

} ;
PTR_LISTS Listes_BA[] = {
       (Ushort)sizeof( L0 ) ,(LIST_ELEMENT*) L0 ,
       (Ushort)sizeof( L1 ) ,(LIST_ELEMENT*) L1 ,

} ;
#define millisecondes *1000
const PTR_INSTRUCTIONS Prg_BA[] = {
       SEND_LIST   , 0 ,
       BA_WAIT     , 5000,
       SEND_LIST   , 1 ,
        TEST_P     , 0 ,
        TEST_P     , 0 ,
       SEND_APER   , 200 millisecondes ,
       SEND_MSG    , 200 millisecondes ,
       BA_WAIT     , 200 millisecondes,
       NEXT_MACRO  , 0

};


B = Bus_Arbitrator__CREATE (
       hdevice ,
       0,          row [0 .. 1]
       0x300,      Label
       Listes_BA,  Name of the table which defines the lists
       Prg_BA      Name of the table which defines the macrocycle
       );
```

16. kfdm_ba_load_macrocycle_fipconfb()

Corresponds to FDM function:

`fdm_ba_load_macrocycle_fipconfb()`

Prototype:

```
BOOL   kfdm_ba_load_macrocycle_fipconfb (
       HANDLE                    hdevice ,
       int                       BA_RANG ,
       const unsigned short *    ba_fipconfb
       ) ;
```

Description:

Used to create and load a macrocycle.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`BA_RANG:`	row of the macrocycle [0 .. 1].
`ba_fipconfb:`	pointer to a table generated in the FIPCONFB tool format. Refer to **[1]**.

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

17. kfdm_ba_delete_macrocycle()

Corresponds to FDM function:

`fdm_ba_delete_macrocycle()`

Prototype:

```
BOOL   kfdm_ba_delete_macrocycle (
       HANDLE      hdevice ,
       int         BA_RANG ) ;
```

Description:

Used to delete a macrocycle.

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`BA_RANG:` row of the macrocycle [0 .. 1].

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

18. kfdm_ba_start()

Corresponds to FDM function:

`fdm_ba_start()`

Prototype:

```
BOOL    kfdm_ba_start(
        HANDLE      hdevice ,
        int         BA_RANG);
```

Description:

Used to start up a macrocycle.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`BA_RANG:`	row of the macrocycle [0 .. 1].

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

19. kfdm_ba_stop()

Corresponds to FDM function:

`fdm_ba_stop()`

Prototype:

```
BOOL    kfdm_ba_stop(
        HANDLE      hdevice);
```

Description:

Used to stop the current macrocycle.

Input parameters:

`hdevice:` HANDLE type reference on the open device.

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

20. kfdm_ba_status()

Corresponds to FDM function:

`fdm_ba_status()`

Prototype:

```
BOOL    kfdm_ba_status (
        HANDLE                  hdevice,
        K_BA_INF_STATUS         Resultats);
```

Description:

Used to read the status of the BA function of the station.

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`Resultats:` pointer to a K_BA_INF_STATUS type structure described below and containing the status of the BA.

```
Typedef struct {
int                     Last_Error;
BA_INF_STATUS           BA_Infos
} K_BA_INF_STATUS
```

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

21. kfdm_ba_commute_macrocycle()

Corresponds to FDM function:

`fdm_ba_commute_macrocycle()`

Prototype:

```
BOOL    kfdm_ba_commute_macrocycle(
        HANDLE      hdevice ,
        int         BA_RANG);
```

Description:

Used to switch a macrocycle.

Input parameters:

`hdevice:`	HANDLE type reference on the open device
`BA_RANG:`	row of the macrocycle [0 .. 1].

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

22. kfdm_ba_set_priority()

Corresponds to FDM function:

`fdm_ba_set_priority()`

Prototype:

```
BOOL    kfdm_ba_set_priority(
        HANDLE                  hdevice ,
        unsigned char           Priority_Level);
```

Description:

Used to modify the priority of the local BA function.

Input parameters:

`hdevice:`	HANDLE type reference on the open devices.
`Priority_Level:`	desired level of priority [0 .. 15], with 0 being the maximum priority.

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

23. kfdm_ba_set_parameters()

Corresponds to FDM function:

`fdm_ba_set_parameters()`

Prototype:

```
BOOL    kfdm_ba_set_parameters(
        HANDLE                  hdevice ,
        enum _BA_SET_MODE       BA_Mode
        unsigned char           MAX_Subscriber,
        unsigned char           MAX_Priority);
```

Description:

Used to modify the parameters used in the time delay calculations of the BA function of the station.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`BA_Mode:`	refer to **[1]** (FDM V4 User Manual).
`MAX_Subscriber:`	idem.
`MAX_Priority:`	idem.

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

24. kfdm_read_presence()

Corresponds to FDM function:

`fdm_read_presence()`

Prototype:

```
BOOL    kfdm_read_presence (
        HANDLE                                  hdevice ,
        int                                     subscriber,
        K_FDM_PRESENCE_VAR          *           Data_Buffer,
        LPOVERLAPPED                            InternalInfos,
        LPOVERLAPPED_COMPLETION_ROUTINE         CallBackRoutine
) ;
```

Description:

Used to read a "subscriber present" variable.

Note

This procedure uses [Write/Read]FileEx in Windows NT.

It must be followed by SleepEx or another equivalent procedure (refer to Microsoft SDK documentation).

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`subscriber:` physical address of the subscriber whose presence variable is to be read.

`Data_Buffer:` pointer to a K_FDM_PRESENCE_VAR type data structure described below:

```
Typedef struct {
     int                Last_Error;
     FDM_PRESENCE_VAR   Result
     } K_FDM_PRESENCE_VAR;
```

`InternalInfos:` refer to `fdm_mps_var_read_far()`

`CallBackRoutine:` refer to `fdm_mps_var_read_far()`

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

25. kfdm_read_present_list()

Corresponds to FDM function:

`fdm_read_present_list()`

Prototype:

```
BOOL    kfdm_read_present_list (
        HANDLE                              hdevice ,
        K_FDM_PRESENT_LIST        *         Data_Buffer,
        LPOVERLAPPED                        InternalInfos,
        LPOVERLAPPED_COMPLETION_ROUTINE     CallBackRoutine
) ;
```

Description:

Used to read a "subscriber present list" variable.

Note

This procedure uses [Write/Read]FileEx in Windows NT.

It must be followed by SleepEx or another equivalent procedure (refer to Microsoft SDK documentation).

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`Data_Buffer:` pointer to a K_FDM_PRESENT_LIST type data structure described below containing the value read:

```
Typedef struct {
     int                Last_Error;
     FDM_PRESENT_LIST   Result
     } K_FDM_PRESENT_LIST;
```

`InternalInfos:` refer to `fdm_mps_var_read_far()`

`CallBackRoutine:` refer to `fdm_mps_var_read_far()`

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

26. kfdm_read_report()

Corresponds to FDM function:

`fdm_read_report()`

Prototype:

```
BOOL    kfdm_read_report (
        HANDLE                                  hdevice ,
        int                                     subscriber,
        K_FDM_REPORT_VAR            *           Data_Buffer,
        LPOVERLAPPED                            InternalInfos,
        LPOVERLAPPED_COMPLETION_ROUTINE         CallBackRoutine
) ;
```

Description:

Used to read a subscriber report variable.

Note

This procedure uses [Write/Read]FileEx in Windows NT.

It must be followed by SleepEx or another equivalent procedure (refer to Microsoft SDK documentation).

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`subscriber:` physical address of the subscriber whose report variable is to be read.

`Data_Buffer:` pointer to a K_FDM_REPORT_VAR type data structure described below containing the value read:

```
Typedef struct {
     int                    Last_Error;
     FDM_REPORT_VAR         Result
} K_FDM_REPORT_VAR;
```

`InternalInfos:` refer to `fdm_mps_var_read_far()`

`CallBackRoutine:` refer to `fdm_mps_var_read_far()`

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

27. kfdm_read_identification()

Corresponds to FDM function:

`fdm_read_identification()`

Prototype:

```
BOOL    kfdm_read_identification (
        HANDLE                              hdevice ,
        int                                 subscriber,
        K_FDM_IDENT_VAR           *         Data_Buffer,
        LPOVERLAPPED                        InternalInfos,
        LPOVERLAPPED_COMPLETION_ROUTINE     CallBackRoutine
) ;
```

Description:

Used to read the subscriber identification variable.

Note

This procedure uses [Write/Read]FileEx in Windows NT.

It must be followed by SleepEx or another equivalent procedure (refer to Microsoft SDK documentation).

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`subscriber:` physical address of the subscriber whose identification variable is to be read.

`Data_Buffer:` pointer to a K_FDM_IDENT_VAR type data structure described below containing the value read:

```
Typedef struct {
     int                    Last_Error;
     FDM_IDENT_VAR          Result

} K_FDM_IDENT_VAR;
```

`InternalInfos:` refer to fdm_mps_var_read_far()

`CallBackRoutine:` refer to fdm_mps_var_read_far()

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

Note

The station identification variable (in comparison with the one used in FDM Version 4) is:

Vendor_Name	Field entered during configuration
Model_Name	Field entered during configuration
Revision	Field entered during configuration
Tag_Name	Field entered during configuration
SM_MPS_Conform	Field entered during configuration
SMS_Conform	Field entered during configuration
PMDP_Conform	Field entered during configuration
Vendor_Field	Field entered during configuration

28. kfdm_read_ba_synchronize()

Corresponds to FDM function:

`fdm_read_ba_synchronize()`

Prototype:

```
BOOL    kfdm_read_ba_synchronize (
        HANDLE                              hdevice ,
        K_FDM_SYNCHRO_BA_VAR      *         Data_Buffer,
        LPOVERLAPPED                        InternalInfos,
        LPOVERLAPPED_COMPLETION_ROUTINE     CallBackRoutine
) ;
```

Description:

Used to read the network BA sync variable.

Note

This procedure uses [Write/Read]FileEx in Windows NT.

It must be followed by SleepEx or another equivalent procedure (refer to Microsoft SDK documentation).

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`Data_Buffer:` pointer to a K_FDM_SYNCHRO_BA_VAR type data structure described below containing the value read:

```
Typedef struct {
     int                    Last_Error;
     FDM_SYNCHRO_BA_VAR     Result
} K_FDM_SYNCHRO_BA_VAR;
```

`InternalInfos:` refer to `fdm_mps_var_read_far()`

`CallBackRoutine:` refer to `fdm_mps_var_read_far()`

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

29. kfdm_messaging_fullduplex_create()

Corresponds to FDM function:

`fdm_messaging_fullduplex_create()`

Prototype:

```
BOOL    kfdm_messaging_fullduplex_create  (
        HANDLE                              hdevice ,
        int                                 Rang ,
        FDM_MESSAGING_FULLDUPLEX  *         Data_Buffer
) ;
```

Description:

Used to create a new FULLDUPLEX messaging system context.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`Rang:`	context number for the driver.
`Data_Buffer:`	pointer to an FDM_MESSAGING_FULLDUPLEX type data structure describing the context. Refer to **[1]** for the detail of this structure.

The scored out fields of the structure below are not to be initialised.

```
typedef struct  {
        enum _FDM_MSG_IMAGE                 Position    ;
        struct{
                    void (*User_Msg_Ack) (FDM_MESSAGING_REF*,
                    FDM_MSG_TO_SEND );
                    void*       User_Qid;  /* optional user info */
                    void*       User_Ctxt; /* optional user info */
                    unsigned short          Channel_Nr;
        }sending;
        struct{
                    void(*User_Msg_Rec_Proc)
                    (FDM_MESSAGING_REF*,FDM_MSG_RECEIVED* );
                    void*       User_Qid ;  /* optional user info */
                    void*       User_Ctxt;  /* optional user info */
                    int         Number_Of_Msg_Desc;
                    int         Number_Of_Msg_Block;
        }receiving ;
        unsigned long Local_DLL_Address;
        unsigned long Remote_DLL_Address;
} FDM_MESSAGING_FULLDUPLEX;
```

Note

The message exchange type at the Data Link Level is acknowledged or not acknowledged according to the contents of the address (see Chapter 2, Subsection 1.6. of **[1]**. Two supplementary bits are used. They are bits 24 and 25.

According to their values the interpretation is the following:

- 0: standard (see Chapter 2, Subsection 1.6 of **[1]**).
- 2: not acknowledged
- 3: acknowledged

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

30. kfdm_messaging_to_send_create()

Corresponds to FDM function:

`fdm_messaging_to_send_create()`

Prototype:

```
BOOL    kfdm_messaging_to_send_create  (
        HANDLE                                  hdevice ,
        int                                     Rang ,
        FDM_MESSAGING_TO_SEND     *             Data_Buffer
) ;
```

Description:

Used to create a messaging context for sending.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`Rang:`	context number for the driver.
`Data_Buffer:`	pointer to an FDM_MESSAGING_TO_SEND type data structure describing the context. Refer to **[1]** for details of this structure.

The scored out fields of the structure below are not to be initialised.

```
typedef struct  {
        enum _FDM_MSG_IMAGE                     Position    ;
        struct{
                    ~~void (*User_Msg_Ack) (FDM_MESSAGING_REF*,~~
                    ~~FDM_MSG_TO_SEND );~~
                    ~~void*       User_Qid;  /* optional user info */~~
                    ~~void*       User_Ctxt; /* optional user info */~~
                    unsigned short    Channel_Nr;
        }sending;
        unsigned long                           Local_DLL_Address;
        unsigned long                           Remote_DLL_Address;
} FDM_MESSAGING_TO_SEND;
```

Note

The message exchange type at the Data Link Level is acknowledged or not acknowledged according to the contents of the address (see Chapter 2, Subsection 1.6. of **[1]**). Two supplementary bits are used. They are bits 24 and 25.

According to their values the interpretation is the following:

- 0: standard (see Chapter 2, Subsection 1.6 of FDM User Manual)
- 2: not acknowledged
- 3: acknowledged

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

31. kfdm_messaging_to_rec_create()

Corresponds to FDM function:

```
fdm_messaging_to_rec_create()
```

Prototype:

```
BOOL    kfdm_messaging_to_rec_create  (
        HANDLE                                  hdevice ,
        int                                     Rang ,
        FDM_MESSAGING_TO_REC      *             Data_Buffer ) ;
```

Description:

Used to create a messaging context for receiving.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`Rang:`	context number for the driver.
`Data_Buffer:`	pointer to an FDM_MESSAGING_TO_REC type data structure describing the context. Refer to **[1]** for details of this structure.

The scored out fields of the structure below are not to be initialised.

```
typedef struct  {
        enum _FDM_MSG_IMAGE                     Position    ;
        struct{
                    void(*User_Msg_Rec_Proc)
                    (FDM_MESSAGING_REF*,FDM_MSG_RECEIVED* );
                    void*       User_Qid ;  /* optional user info */
                    void*       User_Ctxt;  /* optional user info */
                    int         Number_Of_Msg_Desc;
                    int         Number_Of_Msg_Block;
        }receiving ;
        unsigned long                           Local_DLL_Address;
        unsigned long                           Remote_DLL_Address;
} FDM_MESSAGING_TO_REC;
```

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

32. kfdm_messaging_delete()

Corresponds to FDM function:

`fdm_messaging_delete()`

Prototype:

```
BOOL   kfdm_messaging_delete  (
       HANDLE      hdevice ,
       int         Rang ) ;
```

Description:

Used to delete a messaging context.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`Rang:`	number of the context to be deleted.

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

33. kfdm_channel_create()

Corresponds to the FDM function:

`fdm_channel_create()`

Prototype:

```
BOOL    kfdm_channel_create  (
        HANDLE                              hdevice ,
        FDM_CHANNEL_PARAM          *        Data_Buffer ) ;
```

Description:

Used to create a messaging channel.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`Data_Buffer:`	pointer to an FDM_CHANNEL_PARAM type structure describing the context. Refer to **[1]** for the detail of this structure.

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

34. kfdm_channel_delete()

Corresponds to the FDM function:

`fdm_channel_delete()`

Prototype:

```
BOOL    kfdm_channel_delete(
        HANDLE                  hdevice ,
        int                     Channel_nr ) ;
```

Description:

Used to delete a messaging channel.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`Channel_nr:`	number of the channel to be deleted.

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

35. kfdm_send_message()

Corresponds to FDM function:

`fdm_send_message()`

Prototype:

```
BOOL    kfdm_send_message (
        HANDLE                              hdevice ,
        int                                 RANG ,
        K_FDM_MSG_TO_SEND          *        Data_Buffer ,
        LPOVERLAPPED                        InternalInfos,
        LPOVERLAPPED_COMPLETION_ROUTINE     CallBackRoutine) ;
```

Description:

Used to request the sending of a message.

Note

This procedure uses [Write/Read]FileEx in Windows NT.

It must be followed by SleepEx or another equivalent procedure (refer to Microsoft SDK documentation).

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`RANG:` row of the messaging context to be used.

`Data_Buffer:` pointer to a K_FDM_MSG_TO_SEND type data structure described below.

The scored out fields are not to be initialised.

```
typedef struct {
     FDM_MSG_TO_SEND    MsgToSend ;
     void               * Irp;
     FDM_MSG_T_DESC     Block;
     unsigned char      Message[256];
} K_FDM_MSG_TO_SEND     ;

typedef struct  _FDM_MSG_TO_SEND {
     struct             _FDM_MSG_TO_SEND *Next;
     struct             _FDM_MSG_TO_SEND *Prev;
     int                              Nr_Of_Blocks;
     FDM_MSG_T_DESC  *  Ptr_Block ;
     unsigned long           Local_DLL_Address;
     unsigned long           Remote_DLL_Address;
     FDM_MSG_SEND_SERVICE_REPORT
     Service_Report ;
     FDM_PRIVATE_DLI_T   Private ;
```

```
} FDM_MSG_TO_SEND ;

typedef struct _FDM_MSG_T_DESC {
     unsigned short          Nr_Of_Bytes;
     unsigned char       *   Ptr_Data;
     struct _FDM_MSG_T_DESC  *  Next_Block;
} FDM_MSG_T_DESC;
```

`InternalInfos:` Internal structure.

The structure must not be modified during execution of the function.

`CallBackRoutine:` Procedure called on receipt of the acknowledgement of the sending of a message or when time delay expires.

The prototype of the `CallBack` function is as follows:

```
VOID WINAPI CallBackRoutine  (
     DWORD                  dwErrorCode,
     DWORD                  FdmReport,
     LPOVERLAPPED           lpOverlapped      ) ;
```

`dwErrorCode:` is equal to 0 if the request has been processed correctly.

`FdmReport:` not used.

`lpOverlapped:` pointer to an OVERLAPPED type structure.

The OVERLAPPED structure contains information used in asynchronous input and output (I/O).

```
typedef struct _OVERLAPPED {
         DWORD  Internal;
         DWORD  InternalHigh;
         DWORD  Offset;
         DWORD  OffsetHigh;
         HANDLE hEvent;
} OVERLAPPED;
```

`hEvent:` contains user information which may be defined during the send request.

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

There is also a deferred report updated by FDM after sending the message. You have to read the following field.

`InternalInfos.InternalHigh`: which is the FDM report of type:

```
struct {
       enum_FDM_MSG_SND_CNF                    //: bits [16..23]
       enum_FDM_MSG_USER_ERROR                 //: bits [8..15]
       enum_FIP_MSG_SND_REP                    //: bits [0..7]
} ;
```

This deferred report can be explored after successful completion of the `sleepEx` function (or another equivalent procedure) or in the `CallBack` routine.

36. kfdm_received_message()

Corresponds to FDM function:

None.

Prototype:

```
BOOL    kfdm_received_message (
        HANDLE                              hdevice ,
        int                                 RANG ,
        K_FDM_MSG_RECEIVED        *         Data_Buffer ,
        LPOVERLAPPED                        InternalInfos,
        LPOVERLAPPED_COMPLETION_ROUTINE     CallBackRoutine) ;
```

Description:

Used to signal the reception of a message.

Note

This procedure uses [Write/Read]FileEx in Windows NT.

It must be followed by `SleepEx` or another equivalent procedure (refer to Microsoft SDK documentation).

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`RANG:` row of the messaging context to be used.

`Data_Buffer:` pointer to a K_FDM_MSG_RECEIVED type data structure described below:

```
typedef struct {
int                     Last_Error ;
int                     Nbr_Of_Purged_Message;
FDM_MSG_RECEIVED    ref_buffer
FDM_MSG_R_DESC      data_buffer
} K_FDM_MSG_RECEIVED    ;
```

`InternalInfos:` Internal structure.

The structure must not be modified during execution of the function.

`CallBackRoutine:` Procedure called on receipt of the acknowledgement of the sending of the message or when the time delay expires.

The prototype of the `callBack` function is as follows:

```
VOID WINAPI CallBackRoutine  (
     DWORD              dwErrorCode,
     DWORD              FdmReport,
     LPOVERLAPPED       lpOverlapped    ) ;
```

`dwErrorCode` is equal to 0 if the request has been processed correctly.

`FdmReport:` not used.

`lpOverlapped:` not used.

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

Usage example:

```
B = kfdm_received_message (
                hdevice ,
                1 ,                 // [0..29]
                &MsgR,

                &gOverLapped,
                IOCompletionRoutine
);
if ( B == FALSE )                   kufdm_format_message(hLib) ;


Dw =  SleepEx(
            6000,                   // time-out interval in
                                    milliseconds
            TRUE ) ;                // return to execute I/O
                                    completion routine if TRUE
```

//// Here, the message is either read or there is a time out.
In that case, a new read request must be made.

37. kfdm_switch_image()

Corresponds to FDM function:

`fdm_switch_image()`

Prototype:

```
BOOL    kfdm_switch_image (
        HANDLE                  hdevice ,
        K_FDM_IMAGE_NR          *Value
        ) ;
```

Description:

Used to request the switching of the image the AE_LEs are operating with.

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`Value:` pointer to a K_FDM_IMAGE_NR type data structure described below:

```
typedef struct {
     int                    Last_Error;
     enum  IMAGE_NR         Image_Value;
} K_FDM_IMAGE_NR ;
```

`Image_Value:` IMAGE_1 or IMAGE_2

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

38. kfdm_get_image()

Corresponds to FDM function:

`fdm_get_image()`

Prototype:

```
BOOL    kfdm_get_image (
        HANDLE                  hdevice ,
        K_FDM_IMAGE_NR          *Value
        ) ;
```

Description:

Used to find out which image the AE_LEs are operating with.

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`Value:` pointer to a K_FDM_IMAGE_NR type data structure described below:

```
typedef struct {
     int                            Last_Error;
     enum  IMAGE_NR          Image_Value;
} K_FDM_IMAGE_NR ;
```

`Image_Value:` IMAGE_1 or IMAGE_2

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

39. kfdm_ae_le_get_state()

Corresponds to the FDM function:

`fdm_ae_le_get_state()`

Prototype:

```
BOOL   kfdm_ae_le_get_state (
       HANDLE                  hdevice ,
       int                     AE_LE_Rang,
       K_FDM_AE_LE_STATE       *Value
       ) ;
```

Description:

Used to identify the operating state of an AE_LE.

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`AE_LE_Rang:` row of the AE_LE.

`Value:` pointer to a K_FDM_AE_LE_STATE type data structure described below:

```
typedef struct {
     int                     Last_Error,
     enum _FDM_AE_LE_STATE   AE_LE_State;
} K_FDM_AE_LE_STATE ;
```

Report:

This function returns a report with the Boolean value:

TRUE if the request has been processed correctly.

FALSE if it has not.

40. kfdm_generic_time_initialize

Corresponds to the FDM function:

`fdm_generic_time_initialize()`

Prototype:

```
BOOL   kfdm_generic_time_initialize (
       HANDLE hdevice ,
       const FDM_GENERIC_TIME_DEFINITION  *User_param
);
```

Description:

Initialisation of the time management function.

Input parameters:

`hdevice`: HANDLE type reference on the open device.

`User_param`: pointer to an FDM_GENERIC_TIME_DEFINITION structure defined below. The scored out fields are not to be initialised

```
typedef struct {
      enum FDM_BOOLEAN        With_Choice_Producer;
      enum FDM_BOOLEAN        With_MPS_Var_Produced;
      enum FDM_BOOLEAN        With_MPS_Var_Consumed; // FALSE
      enum _FDM_MSG_IMAGE     Image ;                // 1&2

      Ulong                   Refreshment;           // 1 sec
      Ulong                   Promptness;            // -
      const Ulong       *Delta_Time_Location  ;      // interne
      void  (*User_Signal_Mode) ( const int sens );  // interne
      Ushort                  Ticks_Election ;       // 1
      Ushort                  Channel_Nr;            // 1

}  FDM_GENERIC_TIME_DEFINITION;
```

Report:

This function returns a report with the Boolean value:

TRUE: if the request has been processed correctly.

FALSE: if it has not.

41. kdfm_generic_time_set_priority

Corresponds to the FDM function:

`fdm_generic_time_set_priority`

Prototype:

```
Bool
kfdm_generic_time_priority (Handle hdevice, GT-PRIORITY priority );
```

Description:

Used to modify the subscriber priority that is involved in the election process of the time variable producer.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`priority`	value of the priority to set [0..15]. 0 is the greatest priority.

Report:

This function returns a report with the Boolean value.

TRUE: if the request has been processed correctly.

FALSE: if has not.

42. kfdm_generic_time_set_candidate_for_election

Corresponds to the FDM function:

`fdm_generic_time_set_candidate_for_election()`

Description:

Used to include or not include the subscriber in the election process of the time variable producer.

Input parameters:

`hdevice:`	HANDLE type reference on the open device.
`val:`	if **FDM_FALSE**: the subscriber will not be included.
	FDM_TRUE: the subscriber will be included.

Report:

This function returns a report with the Boolean value:

TRUE: if the request has been processed correctly.

FALSE: if it has not.

43. kfdm_generic_time_get_election_status()

Corresponds to the FDM function:

None

Prototype:

```
Bool
kfdm_generic_time_get_election_status (Handle hdevice, Etat Abonnés*
result)
```

Description:

Used to know the status of the subscriber regarding the process of the time variable election.

Input parameters:

`hdevice:` HANDLE type reference on the open device.

`result:` pointer to a `Etat Abonnes` type data structure, described below, that contains the status

```
type def struct{
unsigned                    abonnés: 8;
enum subscriber states      state: 4;
unsigned                    Prio: 4;
} Etat Abonnés
```

Report:

This function returns a report with the Boolean value:

TRUE: if the request has been processed correctly.

FALSE: if it has not.

44. kfdm_generic_time_delete

Corresponds to the FDM function:

`fdm_generic_time_delete()`

Prototype:

```
BOOL    kfdm_generic_time_delete (
       HANDLE hdevice ,
       );
```

Description:

Stops the time management function.

Input parameters:

`hdevice`: HANDLE type reference on the open device.

Report:

This function returns a report with the Boolean value:

TRUE: if the request has been processed correctly.

FALSE: if it has not.

45. kufdm_get_subscriber_number

Corresponds to the FDM function:

None.

Prototype:

```
BOOL      kufdm_get_subscriber_number (
                  HANDLE        hdevice,
                  USHORT   *subscriber);
```

Description:

Utility used to read the number of the station subscriber. The value read is the one contained in the "Registry" which is independent of the value registered by the driver.

Input parameters:

`hdevice`: HANDLE type reference on the open device.

`subscriber`**:** address where the subscriber number will be filled.

Report:

This function returns a report with the Boolean value:

TRUE: if the request has been processed correctly.

FALSE: if it has not.

Chapter 5 Examples of Usage

One example is given on the CD-ROM (running with two subscribers on the same PC):

- production and consumption of MPS variables,
- production and consumption of messages,
- synchronisation of the application with a synchro identifier,
- the subscribers will be bus arbitrators.

An executable file must be created ("console application" type) using the files on the directory.

AELE	Set of globally managed communications variables
API	Application Programming Interface
FDM	FIP DEVICE MANAGER software / library
FIELDRIVE	ASIC / transceiver for copper physical layer
FIELDTR	Isolating transformer
FIELDUAL	ASIC / FULLFIP2 peripheral device for managing medium redundancy
FULLFIP2	ASIC / WorldFIP communications coprocessor
THREAD	Windows NT environment task
VCOM	Communication variable as per the WorldFIP protocol

Glossary